# .NET Remoting

Tomáš Wiszczor

Katedra měřicí a řídicí techniky, FEI, VŠB – Technická Univerzita Ostrava
17. listopadu 15, 708 33, Ostrava-Poruba
Tomas.Wiszczor@vsb.cz

**Abstrakt.** Robustní distribuované technologie jsou základem pro vývoj, běh a správu moderních řídicích a informačních systémů. Mezi základní požadavky kladené na tyto systémy jsou: rychlý a transparentní vývoj, flexibilita vůči změnám a rozšířením, jednoduchost nasazení, snadná údržba, upgrade a bezbečnost těchto systémů. .NET Remoting je jedna ze dvou základních technologii na platformě .NET pro výstavbu distribuovaných řešení. Jejím úkolem je zcela nahradit dosavadní řešení COM a DCOM. Tento příspevěk je věnován .NET Remotingu ze strany architektury a vývoje různých jeho řešení.

**Klíčová slova:** kanál, klient, remoting, server.

## 1 Úvod

Základem distribuovaného řešení je založení spojení mezi různými objekty, představující jistou aplikační logiku, v různých aplikačních doménách na stejném výpočetním stroji (počítači), či libovolně v síti (LAN nebo WAN). K tomu, abychom dokázali vytvořit spojení mezi objekty vyskytujícími se v různých aplikačním doménách, potřebujeme velmi dobré znalosti o objektech, o protokolech určujících spojení, o aplikačním programovém vybavení, o konfiguračních nástrojích a souborech apod. Platforma .NET přináší dvě základní technologie pro podporu distribuovaných řešení, které náročné požadavky na tyto řešení drasticky snižují: ASP.NET a .NET Remoting. ASP.NET řešení je mnohdy známo jako XML Web Sevices (webové služby) a opírá se o webový server (dále jen WS). Díky tomu, že běží nad WS postihuje příslušné bezpečnostní požadavky, škálovatelnost a schopnost uzavírat spojení i s jinými řešeními, běžicími na jiných operačních systémech (dále jen OS). Naproti tomu .NET Remoting stojí tiše v pozadí a rozšiřuje řešení ASP.NET. Není závislý na WS, tedy nemusí postihovat bezpečnost a škálovatelnost, zejména pokud nevyužívá možnost spolupráce s WS a je rychlejší než ASP.NET. Jeho hlavní výhoda je v tom, že vše je v rukou tvůrce (programátora) aplikačního řešení.

## 2 Architektura .NET Remotingu

Celá distribuovaná úloha se skládá ze tří účastníků. Prvním je klient, jenž zasílá požadavky, druhým je vzdálený objekt, jenž umí požadavky obsloužit (logika) a třetím je hostující aplikace – server (hostující proto, že hostuje vzdálený objekt), jenž naslouchá přícházejícím požadavkům a předává je vzdálenému objektu. Klientem a

serverem může být libovolný typ aplikace: konzolová aplikace, služba, webová aplikace nebo tradiční Windows aplikace.

Nyní se podívejme na tyto tří učasníky z hlediska vývoje. Začneme vzdáleným objektem.

## 2.1 Vzdálený objekt

Doporučuje se vždy oddělit vzdálený objekt od serveru, jenž bude následně tento vzdálený objekt hostovat. Důvod je ten, že logiku vzdáleného objektu pak může využívat více druhů serverů. Například si můžeme představit vzdálený objekt jako objekt s funkcí kontroly pravopisu. Pokud ho vystavíme přímo do serveru (například textového procesoru), tak jeho funkce bude moci využívat pouze klient – grafická čast – jenž je pro tento server určen. Klient z tabulkového procesoru již funkce kontroly nemůže využít. Pokud ale kontrolu pravopisu oddělíme od serveru a vytvoříme samostatnou knihovnu, tak bude moci tuto knihovnu využít libovolný server, potažmo klient (znovupoužití).

Aby mohl být objekt používán libovolným klientem, musí umět s ním založit komunikaci. .NET framework poskytuje třídu MarshalByRefObject, která zapouzdřuje tuto schopnost. Co tedy potřebujeme, je pouze z této třídy (při vytváření svého vzdáleného objektu) dědit. Následuje příklad v jazyce C#:

```
using System;

namespace WofexRemoteObject
{
       public class Generator:MarshalByRefObject
       {   private int _hmez=10;
           private Random _hodnota= new Random();
           public int Get(){return _hodnota.Next(_hmez);}
           public void SetScale(int m){_hmez=m;}
       }
}
```

Definovali jsme vzdálený objekt Generator, jenž pomocí metody Get() vrácí novou vygenerovanou hodnotu.

## 2.2 Server – hostující aplikace

Jak jsme si již řekli výše, úkolem serveru je naslouchat klientům na jednotlivé požadavky. Aby server mohl vůbec naslouchat, potřebujeme definovat tzv. komunikační kanál. .NET framework poskytuje dva základní typy kanálů a nebrání se jiným. První druh kanálu je tzv. HTTP kanál. Jak název vypovídá, jde o kanál, jenž používá ke své funkci protokol HTTP. V tomto případě se jednotlivé požadavky od klientů (volání metod vzdáleného objektu) a navrátové hodnoty formatují dle protokolu SOAP. Tento protokol se používá pokut využíváme hostování WS nebo pokud vytváříme distribuované řešení na síti typu WAN. Jeho výhodou je, že může

obejít firewall (standardně se totiž povoluje port 80). Druhým typem kanálu, jenž je k dispozici, je TCP kanál. Jde o kanál, který pracuje na protokolu TCP. Formatování je zde binární a tedy značně rychlejší oproti SOAP formě. Nemá zabudované žádné bezpečnostní prvky jako HTTP kanál ve spojení s WS. Používá se při tvorbě distribuovaného řešení na síti typu LAN. Po zvolení kanálu musíme kanál v .NET zaregistrovat. Při registraci říkáme, jaký kanál budeme používat a na jakém portu budeme naslouchat pro požadavky. Registraci kanálu vytvoří .NET framework objekt kanál, jenž se skládá z klientské části a serverové části kanálu. Serverová část naslouchá a klientská část předává požadavky vzdálenému objektu. Při registraci nelze zaregistrovat kanál s portem, na nemž již někdo naslouchá.

Poté když je kanál správně zaregistrovaný, můžeme přistoupit ke konfiguraci vzdáleného objektu. Konfiguraci rozumíme nastavit typ vzdaleného objektu a jeho URI. Typ vzdaleného objektu může být server-activated nebo client-activated. Typ server-activated říká, že vzdálený object je vytvořen a spravován serverem a typ client-activated zase, že vzdálený object je vytvořen na požádání klienta a spravován klientem. Stav mezi jednotlivými voláními je zachován, nemusíme se bát, že druhý klient bez našeho vědomí stav změnil. Server-activated objekt může být buď Singleton (jedna instance pro každý klient – .NET framework zajistí multithreadní přístup, přičemž stav vzdáleného objektu je stejný pro každého klienta) nebo SingleCall (pro každé volání zajistí zvláštní instanci vzdáleného objektu, bezestavový přístup). Zdrojový kód ukazuje možnost servera jako konzolovou aplikaci:

```
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Http;
using WofexRemoteObject;

namespace WofexServer
{class Server
  {[STAThread]
   static void Main(string[] args)
   {   HttpChannel ch = new HttpChannel(7777);
                    ChannelServices.RegisterChannel(ch);
       RemotingConfiguration.RegisterWellKnownServiceType
                   (typeof(Generator),"MujGenerator",
                    WellKnownObjectMode.SingleCall);
       Console.WriteLine("Server je připraven.");
       Console.ReadLine();
   }
  }
}
```

Všimněte si, že jsme potřebovali pro vytvoření serveru dovést náš vzdálený objekt. Server naslouchá na portu 7777. Uri (externí jméno pro vzdálený objekt) je nastaveno na MujGenerator. Server vytváří server-activated objekt v módu SingleCall.

### 2.3 Klient

Klient už jen volá metody vzdáleného objektu. Může nebo nemusí nakonfigurovat svůj kanál. Pokud nekonfiguruje, tak se použije defaultní, jenž je uveden v konfiguračním souboru machine.config. Pokud chceme ale zajistit, aby klient dokázal naslouchat jiným (tvářil se i jako server), musíme i zde nakonfigurovat správny kanál se serverovou a klientskou částí. Klient může vypadat nasledovně:

```
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Http;
using WofexRemoteObject;
namespace WofexClient
{class Klient
  {[STAThread]
   static void Main(string[] args)
   {   Generator n = (Generator) Activator.GetObject(
                     typeof(Generator),
                     "http://localhost:7777/MujGenerator");
       Console.WriteLine("Vygen. hodnota: {0}",n.Get());
       Console.ReadLine();
   }
  }
}
```

Všimněte si, že pro získání vzdáleného objektu, typu server-activated, jsme použili metodu Activator.GetObject(). V případě, že používáme vzdálený objekt typu client-activated, tak se použije metoda Activator.CreateInstance(). Metod k získání vzdáleného objektu různého typu je ale víc.

## 3 Pokročilejší témata

Když jsme definovali vzdálený objekt, tak jsme dědili od třídy MarshalByRefObject. Tím jsme vlastně řekli .NET, že si přejeme používat daný vzdálený objekt "odkazem". Pokud používame jakýkoliv vzdálený objekt volaný okazem, tak se vytvoří na straně klienta proxy objekt příslušného vzdáleného objektu. Klient tedy musí znát vzdálený objekt, musí být přítomný v odkazech projektu a musíme být schopni na něho ukázat (všimněte si, že v kódu u klienta jsme používali interni funkci typeof()). To vše znamená, že v době vývoje klienta potřebujeme nutně knihovnu vzdáleného objektu. To ale nemusí být vždy jednoduché. Představme si, že vytváříme klienta pro získání počasí. Server již běží; jsou zdokumentovány všechny funkce. Firma, jenž takové služby poskytuje, by musela na svých stránkách poskytovat i příslušnou svou knihovnu(y). Ve skutečnosti se to tak nedělá. Abychom získali referenci na vzdálený objekt můžeme použít vestavěný nástroj v .NET soapsuds. Na straně klienta lze pouze zadat soapsuds –url:http://localhost:7777/MujGenerator?wsdl –gc. Tento nástroj vytvoří příslušný soubor (v tomto případě v jazyku C#). Tento

soubor jenom přídáme do projektu. Tím dokážeme vyvořit proxy objekt na straně klienta.

Jakékoliv volání vzdáleného obejktu se směřuje na proxy objekt, ten volání směřuje na speciální objekt formatter, jenž převede volání do streamu dat. Stream dat se pomocí kanálu převede na stranu servera. Tam se zase v objektu formatter stream dat rozbalí, zavolá se příslušná metoda vzdáleného objektu a navratové hodnoty se podobně převedou zpět. Pokud potřebujeme předávat jako parametry naše objekty hodnotou, musíme je uvést s atributem [serializable]. Na straně klienta se v tomto případě nevytvoří proxy objekt, ale objekt se celý uloží do streamu dat. Na straně serveru se rozbalí, vytvoří v paměti a zavolá. Obě instance jsou tedy pak nezávislé. Velké objekty se ale nedoporučeje takhle přenášet přes síť. Je lepší je volat odkazem.

### 3.1 Životnost vzdálených objektů

V technologií COM a DCOM se životnost objektů řídila pomocí počítadla odkazů. Pokud počítadlo bylo nulové, tak reprezentovalo situaci, kdy na obejkt nikdo neukazuje a může se objekt odstranit z paměti. Tato situace nemusela vždy být ideální. Pokud se z důvodu havárie klienta počítadlo nedostalo na nulovou hodnotu, tak se objekt nikdy z paměti neodstranil. Tato situace je zvlášť nepříjemná u serverů, které běží non stop. To ale z toho důvodu a důvodu Garbage Collectoru, jenž nemůže zjistit referenci v jiných aplikačních doménách, na platformě .NET nejde. Proto se zavádí tzv. životnost vzdálených objektů na základě smluv. Třída MarshalByRefObject poskytuje metodu InitializeLifeTimeService(), kde můžeme nastavit počáteční životnost objektů (například v sekundách), dále o kolik se životnost zvýši při zavolání libovolné metody vzdáleného objektu a tzv. sponzorský timeout. Pokud vyprší příslušný smlouva, tak se vzdálený objekt pokusí zavolat sponzora. Když se sponzor neozve do příslušného sponzorského timeoutu, tak se údaj o sponzorovi ze vzdáleného objektu odstraní Sponzor je speciální objekt, jenž umožňuje prodloužit životnost vzdáleného objektu. Zpravidla je definován na klientské straně.

### 3.2 Konfigurace servera a klienta

Představme si situaci, kdy je potřeba u naprogramované distribuované aplikaci změnit naslouchající port. V případě, že bychom postupovali tak, jak jsme si uvedli, tak je potřeba upravit kód serveru a klienta, znovu zkompilovat a spustit. To může být náročný proces a v některých případech těžko řešitelný. Serverem totiž může být poskytovan třetí firmou, která port dle svých potřeb může změnit a klient zatím stále běží u obyčejného uživatele (někde jinde). Uživatel si zásadně změnu sám neprovede. V tomto případě poskytuje .NET tzv. konfigurační soubory. Konfigurační soubor je ve formátu XML a jsou v něm uvedeny všechny údaje ke správnému nakonfigurování kanálů a vzdálených objektů. Je uložen pod stejným jménem jako server nebo klient (i s příponou), s příponou config. Například v našem případě: WofexServer.exe.config. Konfigurační soubor je umístěn ve stejném adresáři jako server (nebo klient). V

případě, že jako server používáme WS, tak konfigurační soubor je umístěn ve virtuálním adresáři vždy se jménem web.config. V kódu serveru a klienta již pracně nekonfigurujeme kanály, či typy aplikaci. Stačí použít metodu RemotinConfiguration.Configure(), kde jako parametr se předá název konfiguračního souboru. Konfigurační soubor může vypadat jako například:

```
<configuration>
	<system.runtime.remoting>
		<application>
			<service>
				<wellknown mode="SingleCall"
type="Generator, WofexRemoteObject" objectUri="MujGenerator" />
			</service>
			<channels>
				<Channel ref="http" port="7777" />
			</channels>
		</application>
	</system.runtime.remoting>
</configuration>
```

### 3.3 Závěr

Cílem článku bylo uvést nezasvěceného čtenáře do technologie .NET Remoting, nikoliv tedy představit roční práci (díky omezení počtu stránek by se nevešla). Samotná technologie není tak jednoduchá a strohá, jak se na první pohled zdá. Obsahuje množství specializovaných tříd pro pokročilou tvorbu zákaznických distribuovaných řešení počínaje vytváření reálných proxy objektů, sledování marshalingu a konče technikami volání klienta se strany serveru.

## Reference


1. Adrian Turtschi and team, *C # .NET Web Developer's Guide*, Syngress, Rockland 2002. ISBN 1-928994-50-4.



**Annotation.**

*.NET Remoting*

Robust distributed technology are based for development, run and administration of modern control and information systems. Fundamental requests are fast and transparent development, flexibility, simplicity for these systems. .NET Remoting technology is one from two basic solutions where it helps with remote control from one application domain to other (in other computer in a net). .NET provides configuration files that are very useful for configuration servers or clients without recompiling. These technology replace prior technology COM and DCOM. The article is introduction to .NET Remoting.